If you are an ASP.NET developer, you’re probably familiar with the codebehind model introduced by Microsoft to separate the presentation from the logic layer. In a codebehind page, execution of code is event-based. You specify blocks of code to execute when a page loads or when a button is clicked.

The APEX logic layer works in a similar fashion. Perhaps the best way to summarize how it works is this: Page execution is segmented into different chronological events—points at which application logic can be inserted. For instance, the back end execution of a typical APEX page is segmented into two major events, Page Rendering and Page Processing, both of which consist of smaller events that can trigger application logic.

An APEX developer can insert application logic using mostly PL/SQL statements and JavaScript, both of which can also be generated using the wizards provided by APEX. (In fact, if you don’t count PL/SQL as “code,” it’s possible to create an entire bookstore or salesforce application from scratch without writing a single line of code!)

APEX is different from traditional development tools in that it is geared at churning out business applications in the shortest amount of time possible. In meeting that goal, APEX enforces a strict framework that expects business applications to be developed in a certain way. For instance, every form is expected to display information, provide client-side validation and server-side validation, and perform back-end processing at specific points in its lifetime. Applications developed on APEX are limited in nature to this type of flow, and trying to write logic to do something outside this norm may not be as easy as it would be using a traditional development tool. This means that you can’t use APEX to create games (not great ones, at least), but you can certainly use it to create business applications—especially database-centric ones—very quickly.

In this chapter, I explore the APEX logic layer in detail and show you how to perform common tasks in APEX.

3-1. Adding Server-side Validation to Your Form

Problem

You are creating a form to handle patient discharges from a hospital. When nurses fill in the form, they are required to enter the Social Security number of the patient. Before the record can be created, the form data must be validated at the back end—the patient’s Social Security number must be checked against a payment history table. If the payment record exists (if the patient has paid for his or her treatment), the discharge is allowed. If not, the discharge form must display an error message.
Solution

Before you can test the solution in this recipe, you will need to set up some sample database and form objects. After that, you will explore how you can add server-side validation to these forms. Let's start by creating the sample objects.

Setting up the Sample Objects

To create the sample objects, please follow these steps:

1. Create the PaymentHistory table shown in Listing 3-1 using the SQL Workshop or PL/SQL.

Listing 3-1. The PaymentHistory Table

```
CREATE table "PAYMENTHISTORY" (  
"SOCIALSECURITYNO" NVARCHAR2(9),  
"AMOUNTPAID" NUMBER(6,2)
)
```

2. Create the patient discharge table shown in Listing 3-2 (with the DISCHARGEID field set as the primary key), then create a database application and set up a data entry form on top of this table.

Listing 3-2. The PatientDischarge Table

```
CREATE table "PATIENTDISCHARGE" (  
"DISCHARGEID" NVARCHAR2(10),  
"PATIENTNAME" NVARCHAR2(255),  
"PATIENTSOCIALSECURITYNO" NVARCHAR2(9),  
"DATEOFDISCHARGE" DATE,  
"DISCHARGEREMARKS" NVARCHAR2(2000),  
constraint "PATIENTDISCHARGE_PK" primary key ("DISCHARGEID")
)
/
CREATE sequence "PATIENTDISCHARGE_SEQ"
/
CREATE trigger "BI_PATIENTDISCHARGE"  
before insert on "PATIENTDISCHARGE"  
for each row  
begin  
if :NEW."DISCHARGEID" is null then  
select "PATIENTDISCHARGE_SEQ".nextval into :NEW."DISCHARGEID" from dual;  
end if;  
end;
/